(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organizati n
International Bureau

(43) International Publication Date
4 January 2001 (04.01.2001)

PCT

(10) International Publication Number
WO 01/01292 A2

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/EP00/05769

(22) International Filing Date: 22 June 2000 (22.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
9914927.0 26 June 1999 (26.06.1999) GB

(71) Applicant: KONINKLIJKE PHILIPS ELECTRONICS N.V. [NL/NL]; Groenewoudseweg 1, NL-5621 BA Eindhoven (NL).

(72) Inventor: MORRIS, Steven; Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(74) Agent: WHITE, Andrew, G.; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(81) Designated States *(national)*: BR, CN, IN, JP, KR, MX, PL.

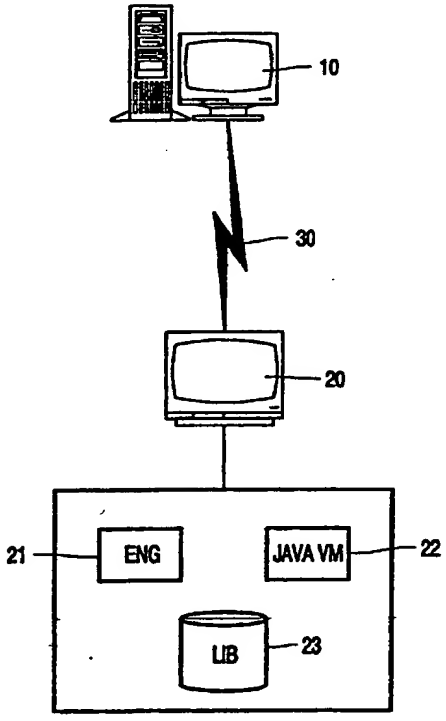
(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: COMPUTER SYSTEM AND METHOD FOR LOADING APPLICATIONS

(57) Abstract: A method and system for loading object-oriented applications comprising the steps of loading the application, determining the class type of each object required by the application, loading the class type(s), including a number of associated functions, required by the application, determining the associated functions loaded with each class, determining the associated functions to be used by the application and removing from memory the associated functions loaded but which are not required by the application.



WO 01/01292 A2

COMPUTER SYSTEM AND METHOD FOR LOADING APPLICATIONS

The present invention relates to a computer system and method for loading applications. In particular, the present invention relates to a computer system and method for application loading in an MHEG-6 environment.

Originally, multimedia was only one of many elements of the World Wide Web. It was never meant as a support technology for the presentation of technical content or for interaction with the outside world. The advent of technologies such as Java (® Sun Microsystems Corporation) and ActiveX has increased the functionality of multimedia and, in turn, the World Wide Web but these have in turn made access to the World Wide Web a memory and CPU resource hungry activity. In a desire to produce low-cost graphical terminals for accessing the World Wide Web, the MHEG (Multimedia and Hypermedia Experts Group) standard was developed to define a basic GUI (Graphical User Interface) and multimedia library for use in kiosk information systems, TV sets, or in video on demand servers for use in communicating with set-top boxes.

The MHEG standard was developed to support the distribution of interactive multimedia applications in a client/server architecture across platforms of different types. MHEG level 5 defines a final form representation for applications such that the appearance of an application is common across all platform types. It allows GUI and multimedia applications to be simply created by defining its constituent components. It is the responsibility of the client terminal to have a run-time system that interprets MHEG commands, to present the application to the user and to handle local interaction with the user. MHEG level 5 is limited in that it does not cater for interaction with the outside world, for example data exchange or the use of client/server applications. MHEG level 6 is in extension to MHEG level 5 that defines a set of mechanisms allowing MHEG applications and Java applications to inter-operate. Java applications are not restricted to multimedia, GUI's and user interaction, offering communication with other computer hosts, applications etc. The inter-operation between Java and MHEG permit MHEG applications to communicate with the external

environment. MHEG I v I 6 is mainly intended to support distribution of interactive retrieval (client/server) applications running on limited resource terminals. The Digital Audio-Visual Council (DAVIC) has adopted MHEG level 6 as the basis for the specification of set-top units for interactive Television.

MHEG applications are delivered to the user as a series of MHEG objects which are interpreted on the user's terminal by an MHEG engine responsible for turning the MHEG objects to graphical and multimedia output at the terminal. Each MHEG level is built on the preceding level. An MHEG engine capable of running an MHEG level 6 application comprises an MHEG level 5 engine interfaced with a Java Virtual Machine. The interface between the Java virtual machine and the MHEG environment allows MHEG applications to run and interact with Java applications.

Java applications are written in terms of a series of objects, each object defining a particular data structure or user interface element of the Java application. Each object is created from a predefined class which defines necessary data structures and functions for an object created from the class to be manipulated or interacted with. These functions are known as methods. A class is effectively a template which is referenced when initially creating an object to determine the structure of the object, the data structures the object may need to operate and also the methods the object can use. When an application is loaded, the classes of objects it uses are checked against those, if any, already held in memory. Any classes that are required but not held in memory are loaded into memory. For each class used to create objects by a Java application, all the methods associated with the class are loaded by the Java virtual machine which runs the Java application. This approach, whilst ensuring that all required methods are loaded, is very wasteful in that many classes contain methods which are not used by the application but which are loaded anyway.

In an environment such as an MHEG-based set-top box, memory is very limited and any memory that can be saved by an application is desirable. The present invention provides a method and system for removing redundant methods from MHEG applications.

According to a first aspect of the present invention, there is provided a method for loading object-oriented applications comprising the steps of loading the application, determining the class type of each object required by the application, loading the class type(s), including a number of associated functions, required by the application, determining the associated functions loaded with each class, determining the associated functions to be used by the application and removing from memory the associated functions loaded but which are not required by the application.

According to a second aspect of the present invention, there is provided a computer system adapted for loading applications, comprising an application loader configured to load an application, to determine the class type of each object required by the application and to load the class type(s), including a number of associated functions, required by the application; and a class processor configured to determine the associated functions loaded with each class, determine the associated functions to be used by the application, and remove from memory the associated functions loaded but which are not required by the application.

Examples of the present invention will now be described with reference to the accompanying drawings, in which:

Figure 1 is a block diagram of a computer system adapted for implementing the present invention;

Figure 2 is a flowchart of a method loading algorithm for use in the present invention; and

Figure 3 is a flowchart of another method loading algorithm for use in the present invention.

Figure 1 is a block diagram of a computer system adapted for implementing the method or system of the present invention.

A computer system running MHEG level 6 applications comprises a server computer 10, a client terminal 20 (such as a television set top box) linked

to th server by a data communications link 30 such as a computer network or telephone network. The client terminal 20 includes an MHEG engine (ENG) 21, a Java Virtual Machine (VM) 22 and a core MHEG library (LIB) 23.

On powering up the client terminal 20, the MHEG engine 21 accesses the core MHEG library 23 and loads an MHEG user interface to enable a user to interact with the client terminal 20. The MHEG engine 21 also loads communication routines from the core MHEG library 23 enabling it to communicate with the Java VM 22 and the server computer 10.

An MHEG application (such as an interactive television programme) accessed or requested by the user via the user interface is requested from the server computer 10 by the MHEG engine 21. The server computer 10 passes the MHEG application to the client terminal 20 as a series of MHEG objects. Once all the MHEG objects of the MHEG application are received by the client terminal 20, the MHEG engine 21 parses the objects and runs the application.

The first time an MHEG application requires to run or interact with a Java application, for example if the users interaction with the MHEG application results in a service in the form of a Java application such as video-on-demand being required from a remote server computer, the Java application is loaded as a series of class objects from the server computer 10 in the same manner as an MHEG application described above. In conventional computer systems the Java VM parses and loads the classes of the objects used by the Java application and all the methods associated with the class into the client terminals memory before creating the application objects from the classes and running the Java application.

Figure 2 is a flowchart of a method loading algorithm for use in the present invention. During the parsing and loading of classes and their associated methods discussed above, the methods are cross-referenced with method calls from the Java application and the MHEG application to determine whether they are used and, if not, they are discarded instead of being loaded.

In step 40, a loaded class from which an object is to be created is parsed to determine the methods it includes. The name or an identifier of each method included in the class is recorded in a table in step 50. In step 60, the method calls

of th Java application and the MHEG application are cross-referenced with the table generated in step 50 to determine (at step 69) whether they are calling a method of the class and, if so, the entry in the table for that method is marked in step 70. Once all the method calls have been parsed (determined at step 71), the methods that are not marked in the table are deleted from memory in step 80.

Figure 3 is a flowchart of another method loading algorithm for use in the present invention. During the parsing and loading of a class and its associated method, it is determined that the class is already loaded and is in use by another application that has previously created objects based on that class. Steps 40, 50, 60, 69, 70 and 71 are performed for the application to be loaded as has previously been discussed with reference to Figure 2. The steps are then also repeated for each further application using the class prior to step 80 to determine the methods they may require. The methods required by the already running application will already be in memory. However, methods not required will have been discarded previously and, if the new application requires any of the discarded methods, the whole class will be reloaded and the set of methods required by both applications is retained in memory.

Whilst the table is normally discarded once redundant methods have been determined, it could be retained so that if another application requires the class, only the methods in addition to those loaded need be determined and loaded with reference to the table.

As an alternative to determining redundant methods whilst the application is being loaded, this process could be performed at the server so only the necessary code is transferred from the server to the client.

The method and system for determining redundant methods according to the present invention could also be called at intervals during the running of the applications to determine methods that are no longer needed.

Although the above description has been mainly concerned with the removal of redundant methods from MHEG applications, the present invention is also applicable to other applications in which portions of possibly redundant code must be loaded, for example Java code can be loaded in one 'chunk' using a

JAR fil to load the data (a JAR file is a collection of Java class files m rged into a single file).

CLAIMS

1. A method for loading object-oriented applications comprising the steps of:
loading the application;
determining the class type of each object required by the application;
loading the class type(s), including a number of associated functions, required by the application;
determining the associated functions loaded with each class;
determining the associated functions to be used by the application; and
removing from memory the associated functions loaded but which are not required by the application.

2. A method according to claim 1, in which in determining the associated functions loaded with the class type(s), a data structure is generated comprising an entry for each associated function loaded.

3. A method according to claim 2, in which in determining the associated functions to be used by the application, the respective entry in the data structure is marked.

4. A method according to claim 3, in which associated functions corresponding to entries not marked in the data structure are removed from memory.

5. A method according to any preceding claim, in which the associated functions comprise methods.

6. A computer system adapted for loading applications, comprising an application loader configured to load an application, to determine the class type of each object required by the application and to load the class type(s), including a number of associated functions, required by the application; and a class processor configured to determine the associated functions loaded with

each class, determin the associated functions to be used by the application, and remove from memory the associated functions loaded but which are not required by the application.

7. A computer program for executing the method of any one of claims 1 to 5.

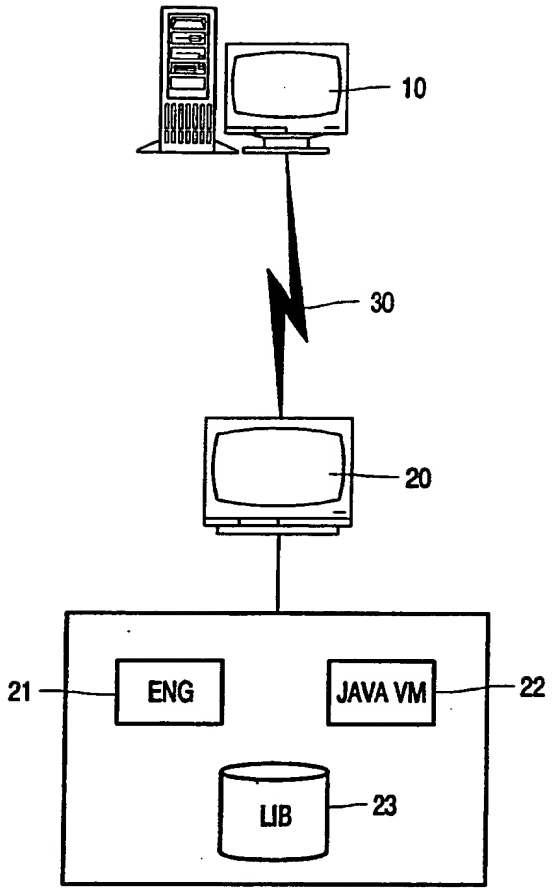


FIG. 1

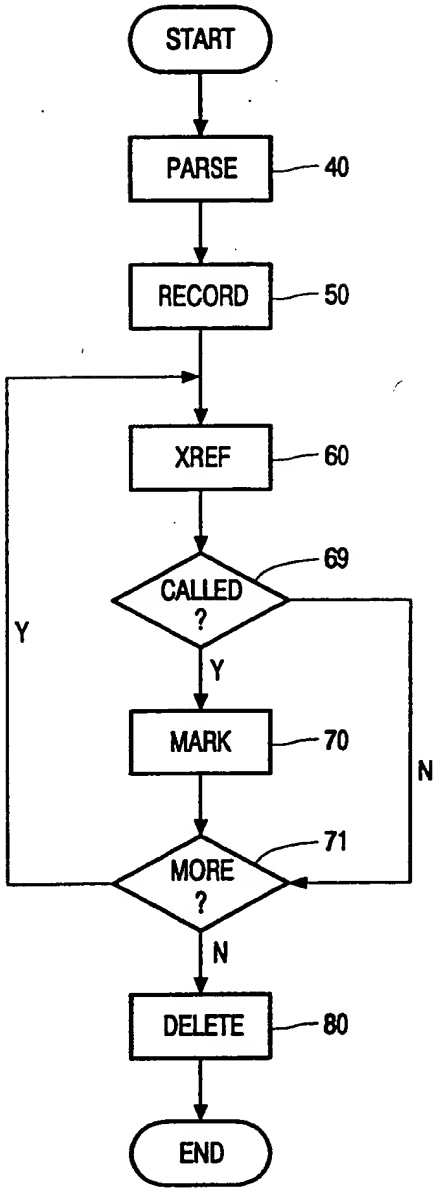

START
PARSE
40
RECORD
50
XREF
60
CALLED ?
69
Y
Y
MARK
70
N
MORE ?
71
N
DELETE
80
END

FIG. 2

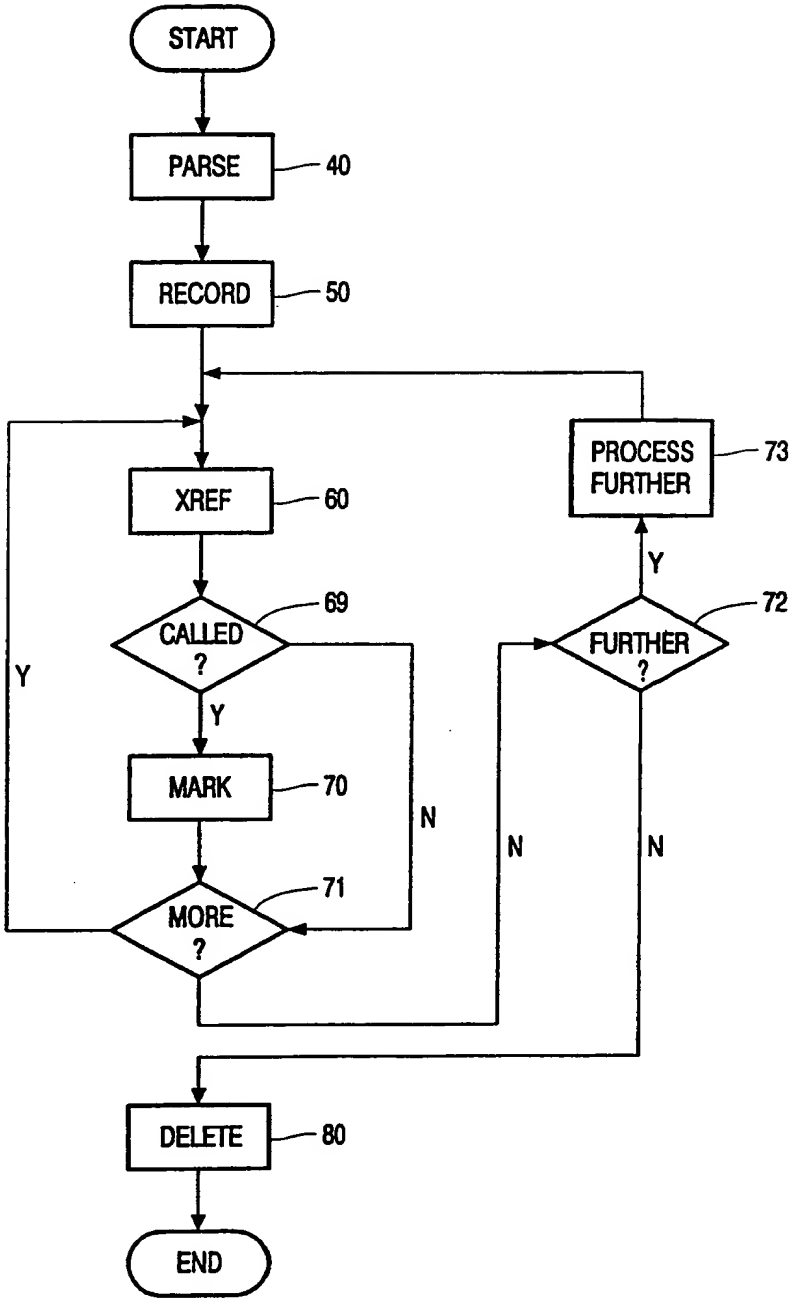

START
PARSE
40
RECORD
50
XREF
60
CALLED ?
69
Y
MARK
70
MORE ?
71
N
Y
PROCESS FURTHER
73
Y
FURTHER ?
72
N
N
DELETE
80
END

FIG. 3